A Language for Construction of Belief Networks

Robert P. Goldman and Eugene Charniak

Abstract—**We describe a method for incrementally constructing belief networks, which are directed acyclic graph representations for probability distributions. We have developed a network-construction language (FRAIL3), which is similar to a forward-chaining language using data dependencies but has additional features for specifying distributions. A particularly important feature of this language is that it allows the user to conveniently specify conditional probability matrices using stereotyped models of intercausal interaction. Using FRAIL3, one can define parameterized classes of probabilistic models. These parameterized models make it possible to apply probabilistic reasoning to problems for which it is impractical to have a single large, static model.**

I. Introduction

UNTIL RECENTLY, probabilistic methods have been considered too unwieldy for use in AI applications. Detractors have cited both the computational complexity of operations over large joint probability tables and the difficulty of assessing or learning the numbers to fill such tables. Recent advances in the use of graphical models for probability distributions, notably belief networks and influence diagrams, have used conditional independences to alleviate these problems.

These graphical formalisms have their drawbacks, however. They are only suited to problems that can be represented in terms of a single probability distribution. Furthermore, their expressive power is only propositional, rather than first order. For this reason, we have developed a language that allows a programmer to specify an entire family of probability models in the form of a set of rules for constructing belief nets. The technique of expressing knowledge about probabilistic models in a knowledge base (KB) and then drawing models from the KB to handle particular problem instances has been dubbed *knowledge-based model construction*. For a review of this topic, see [48].

There are a number of applications where such a facility can be used. Consider the problem of genetic counseling. What one would like to do is draw up a probability model that could be used to assess the chance that the child of some particular parents will have a genetically transmitted disorder. This problem is not amenable to representation in the form of a single belief network model because every family tree (pedigree) is different. What one would like to do is to create, for any given couple, a model that reflects their particular pedigree. Our language provides a convenient facility for combining general knowledge from a domain theory (in this case, genetics) with facts about a specific case (an individual pedigree). We will return to the example of genetic counseling later in this paper because it is well suited to demonstrate features of our language.

Another possible application is the diagnosis of machine faults in a class of machines all built out of a given set of components. For example, one could define a class of models for diagnosing mechanical problems in motorcycles. This class of models would be defined in terms of the operation of components. To create an instance of this class of models for a particular motorcycle, one would specify the values of parameters like fuel supply (carburetion or fuel ignition), transmission (chain or shaft), number of cylinders, etc.

Our technique is tailored to the production of belief networks that can be incrementally constructed and evaluated. Our interest in belief network models grew out of an application in the area of natural language processing [26]. In this domain, any static model will be impractically large—aside from any other issues, just consider the number of words in the English language. For computational practicality, it was necessary that we be able to confine our attention to small, directly relevant portions of such networks. For such situations, our language provides the ability to compose different probabilistic influences in stereotyped ways.

In the next section of this paper, we will give a brief introduction to belief networks. Then, we will give an outline of our language for belief network construction. We will illustrate the use of our language with a simple example from the domain of genetics. Then, we will outline the application of dynamically constructed belief networks to the problem of natural language understanding and the way it has directed the evolution of our network-construction language. Finally, we will discuss related work and future research directions.

II. Belief Networks

Belief networks are directed acyclic graphs (DAG's) used to represent probability distributions. They have been referred to by many other names, including Bayesian networks, directed Markov fields, and causal probabilistic networks. In this section, we give a brief introduction to belief networks. Readers familiar with belief networks may safely skip to Section III. Readers wishing more information are directed to [4], [39], or [37].

The nodes of a belief network represent random variables. For the purposes of this paper, we will be confining our

Manuscript received October 1991; September 1992. This work has been supported by the National Science Foundation under grant IRI-8911122, the Office of Naval Research under grant N00 014-88-K-0589, and a fellowship from Tulane University's Senate Committee on Research. R. Goldman was supported by Rockwell International Science Center, Palo Alto Laboratory, while revising this work for publication.

R. Goldman is with the Computer Science Department, School of Engineering, Tulane University, New Orleans, LA 70118

E. Cherniak is with the Department of Computer Science, Brown University, Providence, RI 02912.

IEEE Log Number 9206607.

0162–8828/93$03.00 © 1993 IEEE

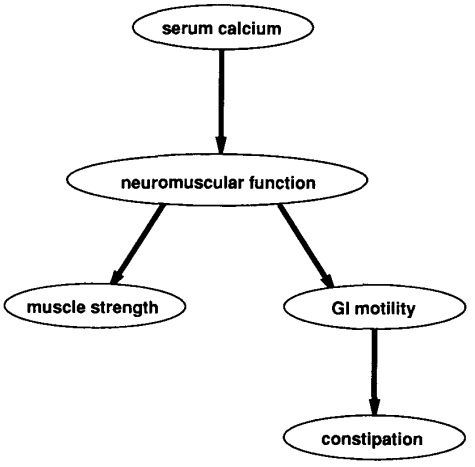


Fig. 1. Fragment of the belief network representation of a medical diagnosis problem.

discussion to discrete random variables with finite sample spaces, but that is not a limitation of the belief network representation. The problems we treat in this paper are all amenable to representation in this form, and the algorithms we use for evaluating such networks are best suited to networks of such random variables.

The arcs of a belief network represent direct influences between random variables. It is conventional, but not necessary, to assign direction to the arcs in agreement with intuitions about causality. For example, Fig. 1 (excerpted from a belief network in [15]) represents a medical diagnosis problem. There are five random variables:

1) serum calcium
2) neuromuscular function
3) muscle strength
4) GI motility
5) constipation.

This diagram represents the *qualitative* information that serum calcium influences neuromuscular function, that neuromuscular function influences muscle strength and GI motility, and that GI motility influences constipation. Furthermore, it represents the assumptions that muscle strength is conditionally *independent* of serum calcium level, given neuromuscular function, that constipation is independent of neuromuscular function given GI motility, etc.

Strictly speaking, it is not accurate to say that arcs represent dependence. More accurately, in a belief network, it is the *absence* of arcs that represents conditional independence. However, that distinction will not be important to our discussion here.

Because belief networks allow us to capture information about conditional independence, they make it possible to compactly represent probability distributions. The probability distribution corresponding to a belief network may be specified by giving conditional probability distributions for each node in the network. The conditional probability distribution gives the probability of the random variable taking on each possible value conditioned on each combination of values of its parent nodes. If the node has no parents (a "root" node), one specifies a prior distribution over the values of that node. For example, let us assume that in the graph given as Fig. 1, serum calcium (SC) was a continuous random variable discretized into $\{high, middle, low\}$, and neuromuscular function (NMF) took on the values $\{normal, impaired\}$. In order to specify a probability distribution for these two nodes, one would need the following:

p(SC = $high$) p(SC = $middle$)
p(SC = low)
and
p(NMF = *normal* | SC = *high*)
p(NMF = *normal* | SC = *middle*)
p(NMF = *normal* | SC = *low*)
p(NMF = *impaired* | SC = *high*)
p(NMF = *impaired* | SC = *middle*)
p(NMF = *impaired* | SC = *low*).

In most situations, this local representation is an immense savings in storage over a table of joint probabilities. It also makes it more convenient to construct stochastic models. This is particularly true in cases where one must rely on expert judgments. Once one has constructed a topological model of a process, it is easier to determine, or subjectively assess, conditional probabilities involving only a small number of variables than to assess the probability of an entire state of affairs (a joint probability).

The local representation also makes possible efficient computation of conditional probability distributions. For example, given that we have observed a patient with constipation and loss of muscle strength, we might want to determine the probability that he/she has an abnormally high serum calcium level.

In general, the problem of finding a conditional probability distribution (we will often use the term "evaluating,"or "solving," the network for the sake of brevity) is NP-hard [14]. However, there are a number of algorithms that are quite effective. Pearl and Kim have developed a linear algorithm for evaluating singly connected belief networks [32], and this algorithm has been generalized to apply to multiply connected networks [39]. Lauritzen and Spiegelhalter have developed a method in which a multiply connected belief network can be transformed into an undirected tree of macro nodes and then evaluated [34]. This algorithm has since been further refined by Jensen *et al.* in their work on the HUGIN system [30], [31]. Finally, there are a number of increasingly sophisticated algorithms based on simulating, and sampling from, distributions represented as belief networks [12], [28], [39]. Early versions of these algorithms performed poorly in distributions that contained conditional probabilities very close to zero [13]. However, a recent algorithm developed by Shachter and Peot is able to use heuristic information to achieve faster convergence [41].

Belief networks may be used in probabilistic approaches to AI problems. Given a network like the one in Fig. 1, one might approach problems of medical diagnosis by finding the conditional probability of various diseases or conditions

given the symptoms a patient presents. Later in this paper, we will see problems of genetic counseling and natural language processing that have been attacked by these representations.

III. An Approach to Network Construction

As a representation for probability distributions, belief networks have many advantages. However, for expressing problems, they have the disadvantages of being propositional and of requiring that all possible problems be specified in terms of one large model whose topology (relations between problem elements) remains fixed. This argues for the development of some more expressive notation for describing problems, which may then be translated, as needed, into belief networks.

The approach we have taken is embodied in a network construction language FRAIL3. FRAIL3 builds on accepted AI technology; it is an adaptation of deductive databases with truth/reason maintenance. FRAIL3's deductive database may be structured into a belief network, where its statements correspond to nodes. We adopt Pearl's idea that causal influences may be thought of as being combined in stereotyped ways [39]. Translated into language constructs, this gives a convenient and compact way to specify the quantities needed to specify a belief network. The belief networks constructed in FRAIL3's database may, at the user's discretion, be projected into data structures that may be evaluated.

FRAIL3 has a great deal in common with a deductive database with truth maintenance (see [11] for an introduction, or see [20], [36], and [19]). Indeed, FRAIL3 was built by modifying such a deductive database. Systems like these, which we will refer to as truth maintenance systems (TMS's), combine a logic-programming language with forward and backward chaining and a data dependency, or justification, network. (Strictly speaking, a TMS is only the software for maintaining the data dependency network.)

Backward-chaining rules allow the system to carry out goal-directed inference to respond to a query. For example, one might have a rule stating that in order to answer a query about whether something has feathers, the system should determine whether the thing in question is a bird. Conversely, forward-chaining rules, or production rules, allow programmers to specify that the addition of some propositions to the database can trigger the addition of other statements. For example, one might want a rule that states that whenever one learns of the existence of a bird, one should record the fact that this new bird can fly.

When propositions are added to the database of a TMS, they are given a *justification*, which records the antecedent statements necessary to justify belief in the consequent. These justifications are records that point from the justifiers to the justified statement. To return to the example above, if we had asserted into our database the fact that Tweety is a bird, our forward-chaining rule would cause the addition of the further statement that Tweety can fly. The database would also contain a justification—a record that states that the proposition that Tweety can fly is justified by the fact that Tweety is a bird and by the forward-chaining rule that states that all birds fly. More formally, we may think of a justification as a hyperedge from the antecedent statements to the consequent. A hyperedge is an ordered pair whose first element is a set of vertices (the tails of the edge) and whose second element is a single vertex (the head). Fig. 2(b) contains two hyperedges: $(\{a, b, c\}, v)$ and $(\{d, e\}, v)$. Therefore, we may view a truth maintenance database as a network in which the nodes are propositions, and the (hyper)edges that connect them are justifications.

Belief networks, like data-dependency networks, may be thought of as hypergraphs. In some ways, it is better to view them in this way since the probability distribution of child nodes depends on the *combination* of values of all of its ancestors. When one takes this view, it is easy to see the parallel between constructing data-dependency nets and constructing belief nets.

FRAIL3 is similar to a deductive database language in having rules for forward and backward reasoning. It differs in two important ways. First, the networks built are not used for truth/reason maintenance. Instead, they may be interpreted as belief networks. To emphasize this distinction, we will refer to our forward-chaining rules as network-construction rules. Second, FRAIL3 allows the rule-writer greater flexibility in specifying the way arcs are added. A TMS always adds hyperedges (justifications) from antecedent statements to consequents. FRAIL3 allows arcs to be added in arbitrary directions as long as the resulting network contains no cycles.

We need to be able to specify directionality in the rules because the rules have two different readings: declarative and procedural. The declarative reading is as information about probabilistic influence. The procedural reading is as a rule for constructing a fragment of a belief network. As far as the declarative reading is concerned, it is customary, when formulating a belief network, to direct the arcs in accordance with causality. To return to the example of Fig. 1, it would be unusual to direct the arcs from symptoms (e.g., constipation) to conditions (serum calcium level). Pearl argues for directing arcs from causes to effects in his book [39]. The causal interpretation provides a good method for ensuring that the network will be a DAG and simplifies assessment of the conditional probabilities.

When rules are used as procedures for network construction, on the other hand, we often need a different directionality. In diagnostic and interpretive applications, one often needs to reason from effects to causes. The observation of *effects* (the variables that will eventually be at the heads of arcs in the belief network) will be used as triggers of rules that construct these networks. To sum up, our knowledge about how to construct a model to explain an observation is most naturally directed from effects to causes; the knowledge captured by the resulting model is most naturally directed from causes to effects.

The data structures in FRAIL3 that correspond to data-dependency records in TMS's are called "probability-forms" (pforms). Each pform is a hyperedge from a set of statements to one statement. This hyperedge corresponds to a set of edges in a belief network; see Fig. 2. Pforms differ from justifications in that they contain information used to construct the conditional probability distributions of their child nodes.

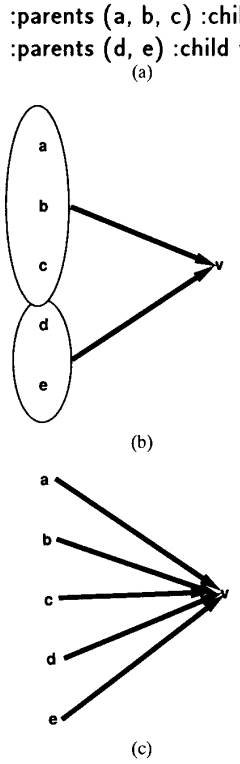


Fig. 2. Interpretation of pforms: (a) pforms; (b) coresponding hypergraph; (c) corresponding belief network.

proposition	::=	{predicate term*}
term	::=	atom || (function term*) || variable
function	::=	atom
variable	::=	?atom
label-variable	::=	??atom
network-construction rule	::=	(→ trigger consequent [:**prob** pform*])
trigger	::=	proposition [:**label** label-variable]
consequent	::=	{proposition [:**label** label-variable]} || conjunction || →← rule
conjunction	::=	(**and** {proposition [:**label** label-variable]}*)
→← rule	::=	(→← condition consequent [:**prob** pform*])
condition	::=	{proposition [:**label** label-variable]} || conjunction
backward-chaining rule	::=	(← query sub-goal)
query	::=	proposition
sub-goal	::=	proposition || (**and** proposition*)
pform	::=	(edge-spec prob-spec)
edge-spec	::=	label-variable* ⇒ label-variable
prob-spec	::=	(case-spec prob)*
case-spec	::=	(value | value*)
prob	::=	number || :p
value	::=	atom

Fig. 3. BNF syntax chart for FRAIL3. Parentheses, bold-faced words, the question mark, the single vertical bar, →, ←, ⇒ and → ← are tokens. Curly brackets are used for grouping. Square brackets surround optional constituents. A doubled vertical bar is used for disjunction. Kleene star (*) is used to indicate zero or more repetitions of a constituent.

IV. SYNTAX OF FRAIL3

In Fig. 3, we give a BNF representation of the syntax of FRAIL3. For the sake of implementation convenience, we have adapted a Lisp-like syntax. Lisp atoms are used as the names of functions, constants and predicates, and special lisp atoms (beginning with ? or ??) are used to represent logical variables. Propositions (that may also be interpreted as random variables), network-construction rules, and backward-chaining rules are stored in the database of FRAIL3. Some sample propositions are

```
procedure assert (proposition)
   add-to-database(proposition)
   let rules = retrieve-matching-rules(proposition)
      foreach rule in rules do
         let unifier = unify(proposition,antecedent(rule))
            if unifier then
               let new-proposition = apply-bindings(unifier,consequent(rule))
                  assert(new-proposition);
            endif
      od
end assert;
```

Fig. 4. Pidgin Algol for the procedure that adds a statement to FRAIL3's database showing how network-construction rules are applied as a side effect.

(can-fly tweety)
(can-fly (son tweety))

A sample backward-chaining rule is

(← (self-propelled-fly-to ?agent ?destination)
(can-fly ?agent))

which says, roughly speaking, that if you want to know whether an agent can fly itself to some destination, check to see if that agent can fly. We will not spend much time discussing backward-chaining rules; for ways of implementing backward-chaining in such a database, see [11] or [38]. We will discuss the syntax and semantics of the network-construction rules in the following section.

V. RULES IN FRAIL3: NETWORK TOPOLOGIES

One way in which FRAIL3 differs from conventional AI database languages is in the expressiveness of its rule language. FRAIL3 allows the specification (both qualitative and quantitative) of probabilistic dependence between the statements/random variables in its database. In this section, we will show how these dependencies may be expressed, using the two important operators → and →←.

→ is the basic operator used in writing network-construction rules. These rules are of the form (→ *trigger consequent* `:prob` *pforms*) (cf. Fig. 3). As with ordinary forward-chaining systems, when a statement unifying with *trigger* is added to the database, the bindings are applied to *consequent*, and the resulting statement is added to the database; see Fig. 4 for a pseudo-code description of this process.

The *pforms* control the way arcs are added to the network and give information about the way the conditional probability matrices are set up at the tails of the arcs. This information will be used by the distribution functions discussed below.

Consider a sample rule:

Rule 1

(→ (grass-wet ?date) :label ??caused
(rain (night-before ?date)) :label ??cause
:prob ((??cause ⇒ ??caused ((t | t) 0.9) ((t |f) :p))))

This states that whenever we learn that the grass is wet, we should create a random variable for the possible explanation that it has just rained. Therefore, if one asserted to FRAIL3 that

(grass-wet today)

FRAIL3 would create the new statement

(rain (night-before today))

Now that we are able to add statements to the database, it is necessary that we be able to connect them and quantify these connections. Statements are connected by means of `:labels`

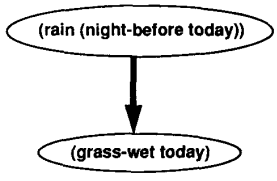


Fig. 5. Database update after sample rule is triggered.

and `:prob` forms. The `:labels` allow us to bind the actual statement data structures (as opposed to the list of atoms, which is the content of the statement) to logical variables. In Rule 1, the logical variable `??caused` is unified with the statement structure that triggers the rule and `??cause` with the statement added as a result of the rule's firing. After the rule is fired and all nodes are added to the database, the forward-chaining routine will interpret the :prob form to add an arc from `??cause` to `??caused`; see Fig. 5. It should be emphasized that unlike a conventional TMS, which automatically adds arcs from antecedents to consequents, FRAIL3 does not add any arcs other than those explicitly designated by `:prob` arguments to rules. Further, although this sample rule only adds a single, simple arc, it is possible for a rule to add more than one arc and to have more than one node at the tail of the arc (i.e., add a hyperedge).

The information used in quantifying the belief network is also given in the `:prob` argument of the → rules. Associated with each hyperedge (pform) is some conditional probability information. In Rule 1, the second part of the `:prob` argument indicates that when we draw up the conditional probability matrix for the `condition` node, its probability given that `causal-event` is true is 0.9. Its probability given that `causal-event` is false should default to the prior probability, that is, the meaning of the `:p` special argument. We will have more to say about quantifying distributions in the next section; for now we will concentrate on adding nodes and edges to the network.

Sometimes simple forward-reasoning rules like the ones discussed above are not sufficient. There are two particularly common cases where this occurs. First, we may wish to consult a KB to determine how to expand the network. For example, in a natural language processing (NLP) application, we may wish to look up possible word meanings in a dictionary when a particular word is observed and proceed accordingly. Second, we may need to take into account what nodes are in the network already when we decide how to expand it. This case also arises in our NLP application. For example, when we observe a pronoun, we create nodes representing various reference hypotheses. The entities to which a pronoun could refer are limited to those previously mentioned in discourse (and, hence, appearing in the belief network). We will return to both of these examples below, and again in Section IX.

The same mechanism—the →← operator—is used to handle both the case of incorporating information from the KB and expansions conditional on previous network contents. →← rules are of the form (→←*condition consequent pforms*) (cf. Fig. 3). →← rules may be used in place of an ordinary consequent in → rules. The way they work is as follows: If

```
else
  add-to-database(proposition)
  let rules = retrieve-matching-rules(proposition)
    foreach rule in rules do
      let unifier = unify(proposition,antecedent(rule))
        if unifier then
          let new-proposition = apply-bindings(unifier,consequent(rule))
            assert(new-proposition);
        endif
    od
  endelse
end assert;

procedure run-→←-rule(pirule) ;partially-instantiated rule
  {first find all ways that the condition can be proven from the KB}
  let all-binds = prove(condition(pirule))
    foreach bind in all-binds do
        let new-proposition = apply-bindings(bind,consequent(pirule))
          assert(new-proposition);
    od
end run-→←-rule;
```

Fig. 6. Pidgin Algol for the procedure that adds a statement to FRAIL3's database. This is a revised version of Fig. 4, showing how →← rules are implemented.

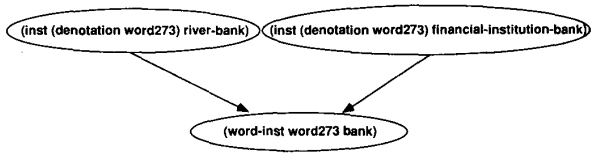


Fig. 7. Network resulting from firing of word-sense rule.

FRAIL3 is told to add a statement whose "predicate" is →←, FRAIL3 will instead attempt to find all possible instantiations of *condition*, apply the resulting bindings to *consequent*, and add all resulting consequents; see Fig. 6 for a revised pseudocode for the assert procedure, showing how →← rules are implemented.

For example, the following is a rule used in our language-understanding application for reasoning about word senses (semantic processing). The problem at hand is the following: We have observed a word in some sentence, and we want to consider hypotheses that would explain the use of this word. The use of the word could be explained by the intent that the word refer to something in the world. A dictionary of possible word senses could be stored in a FRAIL3 KB or, more likely, stored separately and accessed by the FRAIL3 inference engine.

Rule 2

```
(→ (word-inst ?i ?word) :label ??A
    (→← (word-sense ?word ?frame ?prob)
          (inst (denotation ?i) ?frame) :label ??C)
   :prob ((??C ⇒ ??A) ((t | t) ?prob)))
```

This rule may be read as follows: If a node is added to the network describing a new word token (an `instance` of a word), and if there is a statement in the database specifying that one of the senses of this word is `?frame`, add a node for an instance of the type `?frame`. Draw an arc from this newly added node `(??C)` to the `word-inst` node `(??A)`. When drawing up the probability matrix for the `word-inst` node, use information about the frequency with which it is used to express the concept `frame`. For example, if the word encountered is "bank," there will be two possible senses: river-bank and financial-institution-bank. Therefore, there would be two ways to satisfy the additional query: (word-sense ?word ?frame ?prob), with ?frame bound to river-bank and financial-institution-bank. Each of those would cause the creation of a new hypothesis. The resulting network is shown in Fig. 7.

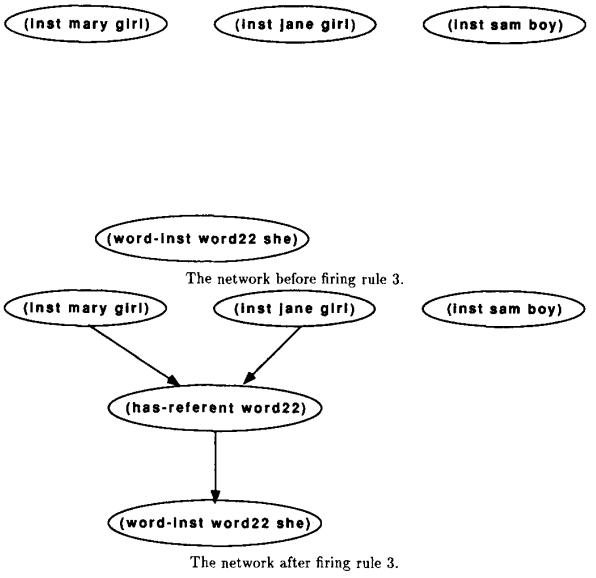


Fig. 8. Belief network before and after firing pronoun referent rule. Note that Sam, not being female, is not considered a possible referent for the pronoun.

We also give an example of the use of →← to conditionalize rules on the current belief network. As mentioned above, one of the tasks for our natural language system is to find referents for pronouns. A desire to refer to an already-discussed person or thing can be the explanation for using a pronoun. Therefore, when we see a pronoun, we need to find all the previously known entities that are of the same gender as the pronoun and build arcs from them to the newly added statement. This requires the kind of combined forward and backward chaining that →← provides. A simplified version of our pronoun rule follows:

Rule 3

```
(→ (word-inst ?i ?pronoun) :label ??PRONOUN
  (→← (and (inst ?x ?frame) :label ??POS-
REFERENT
       (compatible-gender ?pronoun ?frame))
    (has-referent ?i) :label ??HAS-REF)
    :prob (??POS-REFERENT ⇒ ??HAS-REF
      ((t | t) (ref-prob ?frame ?pronoun)
        ((t | f) 0)))
      (??HAS-REF ⇒ ??PRONOUN)
        (((t | t) 1) ((t | f) 0.00001)))
```

Note that this one rule adds more than one arc to the belief network: one from (has-referent ?i) to the node for the pronoun, representing the fact that an intent to refer to something already mentioned can cause the use of a pronoun, and one from each possible referent to the has-referent statement. Fig. 8 gives an example showing the effect of the firing of this rule in response to a use of the pronoun "she" in a context where two female names (Mary and Jane) and one male name (Sam) have been mentioned.

VI. Constructing Probability Distributions

In the previous section, we explained how the language FRAIL3 may be used to lay out the topology of a belief network. Here, we discuss the representation of the conditional probability matrices that are associated with each node. In doing so, we make use of Pearl's suggestion that there are "canonical models of multicausal interactions" (see page 184 of [39]). The FRAIL3 language has features that make it convenient to specify the canonical models that are associated with various types of random variable.

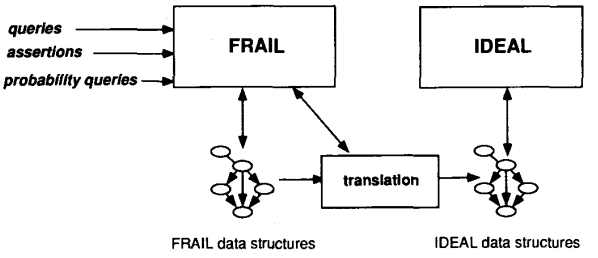


Fig. 9. Architecture of the FRAIL3 system.

In order to understand the mechanics of conditional probability representation, one must know a bit about FRAIL3's architecture; see Fig. 9. As explained above, the statements in FRAIL3's database, which may be accessed by conventional deductive database queries, may be connected into a belief network. In order to find the probability distribution of such a network conditioned on some observations, the network is copied into a set of data structures in a package called **i**nfluence **d**iagram **e**valuation and **a**nalysis in **l**isp (IDEAL) [47]. IDEAL contains implementations of various algorithms for finding posterior distributions ("solving" networks). We will have more to say about the process of evaluation in Section VII. In the meantime, it will be important to keep in mind that conditional probability matrices are represented only *implicitly* in FRAIL3 and expanded fully into arrays only in the corresponding IDEAL data structures. In FRAIL3, conditional probability matrices will be represented as compositions of pforms, prior functions, and causal combination functions. These latter two types of functions are actually not pieces of FRAIL3, per se. Rather, they are part of the component that translates FRAIL3 data structures into IDEAL belief nets for solution. In this section, we will discuss the nature of the abstract representation, and in the following section, we will discuss its use in translating from FRAIL3 to IDEAL.

In the simplest case of network construction, there is exactly one pform for each node, and that pform specifies fully the conditional probability matrix for the node at the head of the corresponding hyperedge. Consider for example, the following rule for the case of a bell that rings if and only if a corresponding coin-flip turns out heads:

Rule 4

```
(→ (bell-rings) :label ??BELL
    (fair-coin-flip) :label ??COIN
    :prob ((??COIN ⇒ ??BELL)
          ((t | heads) 1.0) ((t | tails) 0.0)))
```

This simple example may be used to illustrate the need for *prior functions*. These are used to represent the distribution of a node when that node has no parents and, hence, no pforms (we call such nodes "root nodes"). Consider the case

of `(fair-coin-flip)` in Rule 4. We would like to specify that variables of the type `fair-coin-flip` have the prior distribution of heads 50% and tails 50%. We may write a prior function to record this fact. The prior function is a piece of Lisp code used to fill out an IDEAL conditional probability matrix. Prior functions are associated with various predicates. When a node has no pforms, the translation program will invoke the prior function corresponding to its predicate when it translates that node into an IDEAL data structure. We will see an example of the use of a prior function in Section VIII.

The causal combination functions are more complex. One difficulty in the use of belief networks in AI application is that they require the assessment of quantities that are not easily available to our intuition or are easily assessed from data. These are the probabilities of occurrence of random variables conditioned on *combinations* of values of causes. Pearl gives a clear example:

> . . . in our burglary alarm scenario, it is reasonable to expect people to have some prestored idea of the likelihood that a burglary will trigger an alarm or that an earthquake will trigger it, but not of the likelihood that the combination, burglary and earthquake, will trigger it (see page 184 of [39]).

Pearl argues that people have domain-independent canonical models of intercausal interaction and that these correspond to noisy forms of conventional boolean gates. Because of his interests, he concentrates on the discussion of the "noisy-OR gate." This is used for situations where there are several possible causes of a given piece of evidence, and it is assumed that the various causes do not interfere with each other. He suggests, however, that other patterns of interaction (e.g., conjunctive rather than disjunctive) are also possible.

Like Pearl, we wish to use canonical models for intercausal relations. Therefore, FRAIL3 allows the user to specify what we call *causal combination functions*. Causal combination functions combine the information from various pforms (recall that a random variable will have a pform corresponding to each causal influence). If there is only one pform, there is no need to combine multiple causes, and the single pform will completely specify the node's conditional probability matrix. Among the functions we use as causal combination functions are noisy versions of OR and AND gates and a sort of "noisy multiplexer" we call ONEOF.

These causal combination functions are lisp functions that take a set of pforms as arguments and generate a conditional probability matrix. Let us consider Pearl's burglar alarm scenario again. Assume that we have chosen noisy-OR as our causal combination function, as Pearl suggests. When explicitly specifying the conditional probability matrix for the burglar alarm scenario, the noisy-OR function will have two pforms: one that specifies the chance that the alarm will go off given a burglary is in progress and one that specifies the chance that the alarm will go off given that an earthquake happens. Out of those two pforms, a matrix that contains the probability of burglar alarm given *four* cases must be constructed. These cases are as follows:

1) burglary and no earthquake
2) earthquake and no burglary
3) both earthquake and burglary
4) neither earthquake nor burglary

The noisy-OR causal combination function dictates that we proceed as follows: In the case where only one of the causal factors is true, accept the probability given by that pform. Therefore, for case number 1, we would take P(alarm|burglary). For the case where both of the causal factors are present, assume that they do not interact in any way. Therefore, accept the greater of P(alarm|burglary) and P(alarm|earthquake). Finally, accept a base rate for case 4, where neither of the causal effects is present.

For another example, consider the rule for word-senses, given previously as Rule 2. Let us recall what that rule says: When one encounters a word, look up all its possible meanings. For each possible meaning, create a hypothesis node, and connect that hypothesis node with the node representing the word. Typically, words are ambiguous and have a number of senses. Each such sense will give rise to a different pform. We want our causal combination function to dictate that each one of these senses is a possible explanation for the use of the given word and that the senses are mutually exclusive.

The causal combination function we use for word-inst statements is `oneof-dist`. The gist of what `oneof-dist` does is the following: If exactly one of the parent causes is true, then that cause dictates the chance that the child is true. If more than one of the parent causes is true, then the child is not true. This is our way of ruling out, e.g., more than one possible meaning for a word. Finally, if none of the parent causes obtains, then there is some default chance of the child being true anyway (this is the noise in "noisy-ONEOF").

In constructing FRAIL3, we have chosen to use the predicates of the statements/random variables to control selection of causal combination functions. We use a declaration form we call **defpreddist** to associate a given predicate with the appropriate causal combination function. Therefore, for example, since word tokens are represented by the propositions whose predicate is `word-inst`, we use the form:

```
(defpreddist word-inst oneof-dist)
```

In this section, we have detailed how conditional probability matrices are represented in FRAIL3. We have shown that FRAIL3 allows the specification of prior and causal combination functions. These make it possible to represent the prior probability of random variables and the way that different causes combine to influence them. The choice of an appropriate prior or causal combination function is controlled by associating them with the predicates of FRAIL3 statements. Finally, predicates may have associated default probabilities, which make it more convenient to write the causal combination functions.

VII. How the Networks Are Evaluated

As mentioned in the previous section, we do not directly evaluate the networks in FRAIL3's database. Instead, these networks are first translated into secondary data structures to be evaluated by a separate software package. The translation

```
procedure translate-to-ideal (frail-stms);
var ideal-nodes;

    foreach stm in frail-stms do
       let ideal-node = create-ideal-node(stm)
           add ideal-node to ideal-nodes
    od
    foreach ideal-node in ideal-nodes
              and stm in frail-stms do
       if stm has no parents then
          cond-prob-matrix(ideal-node) = apply prior function(stm) to ideal-node
       else if stm has multiple pforms then
          cond-prob-matrix(ideal-node) = apply causal combination function(stm)
                                              to ideal-node and pforms(stm)
       endif
    od
    return ideal-nodes
end translate-to-ideal;
```

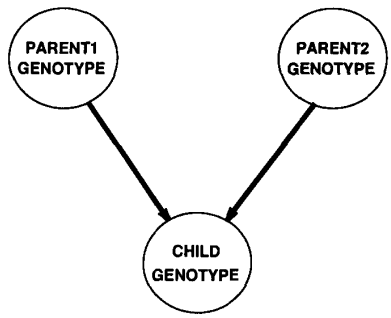
Fig. 10. Pseudo-code for translation module.

takes place in response to a request to find a probability distribution over the belief network represented in FRAIL3's database. It is in the process of this translation that the prior and causal combination functions are invoked. Accordingly, the actual conditional probability matrices are represented only *implicitly* in FRAIL3 in the form of pforms, causal combination functions, prior functions, and default values. For a sketch of this architecture, see Fig. 9. Pseudo-code of the procedure that translates FRAIL3 networks into IDEAL networks is given as Fig. 10.

Currently, we are using the IDEAL system [46] to evaluate the belief networks. IDEAL is a program allowing the specification of belief networks and influence diagrams in LISP. It provides a framework within which a number of network-solution algorithms can be applied to such diagrams. We have written interface functions that first create an image of the FRAIL3 network's topology in IDEAL data structures and then flesh out the corresponding matrices using the techniques discussed above.

The decision to use IDEAL as the evaluation component of FRAIL3 was initially simply a matter of convenience. There have been several happy side effects of the decision to keep the evaluation and construction aspects of FRAIL3 separate. First, the implicit (FRAIL3) representation of the distribution is more efficient than the explicit one used in evaluation, and it is possible for applications to make qualitative decisions based on the more compact representation before more expensive evaluations are done. Second, we can experiment freely with different evaluation algorithms. IDEAL provides a number of such algorithms. There have also been experiments in using some search algorithms and dynamic programming algorithms for processing of networks built using FRAIL3 (see, for example, [44], [21], [10], [43]). We expect to build filters to translate FRAIL3 networks into representations for other belief network packages like CaBEN [16], SPI [18], or HUGIN [1], as such opportunities arise.

Finally, and on a related note, we are able to customize the filter programs that translate to the evaluation structures in order to tune the representation to particular problem solving techniques. For example, when using Jensen's variant of the Lauritzen and Spiegelhalter algorithm [30], [31], we are able to add intermediate nodes and absorb some evidence nodes into their parents. It would not be appropriate to



	a1a1	a1a2	a2a2
a1a1	(1, 0, 0)	(0.5, 0.5, 0)	(0, 1, 0)
a1a2	(0.5, 0.5, 0)	(0.25, 0.5, 0.25)	(0, 0.5, 0.5)
a2a2	(0, 1, 0)	(0, 0.5, 0.5)	(0, 0, 1)

Fig. 11. Conditional probability matrix for transmission probabilities. The three entries in each cell are P(a1a1 | *conditioning-case*), P(a1a2 | *conditioning-case*), and P(a2a2 | *conditioning-case*), respectively. Genotypes of parents are the horizontal and vertical indices.

make such changes in the rule-based network construction process. First of all, these changes are specific to a particular evaluation algorithms. The ones mentioned here, for example, would not be suitable when using a simulation algorithm. The performance of simulation algorithms is more sensitive to features of the belief network conditional probability matrices than to the topology of the networks. The known exact algorithms have the opposite behavior. Building them into the rulebase would make it more difficult to experiment with different algorithms. A second consideration is that making such changes in the network construction rules would be an unwelcome mixture of procedural and declarative issues. It would make the writing of the rules conceptually more difficult.

VIII. AN EXAMPLE FROM GENETICS

To give an idea of how our system can be used, we give our treatment of an example in genetics. We have adapted this example from [31] and [45].[1] This example has been chosen solely for its convenience and clarity.

We consider a gene with two alleles (or values), a1 and a2. Every individual's *genotype* consists of two alleles: one inherited randomly from each parent's pair. Because we have no way, in general, of knowing from which parent a given allele has been inherited, we represent genotypes as unordered pairs: a1a1, a1a2, or a2a2. a1 is recessive and gives rise to a detectable condition; a2 is dominant and does not. Therefore, the *phenotypes* (the detectable conditions) are *present*, corresponding to genotype a1a1, and *absent*, corresponding to genotypes a1a2 and a2a2.

In order to model the situation resulting in populations with these genotypes, we need to know probabilities determining the transmission of the genes from parents to children (these are called *transition probabilities*). These are given in Fig. 11. In order to avoid infinite regress, we must make some assumptions about the genotypes of individuals about whose

[1] Any errors are our fault, not theirs.

ancestry we are ignorant. These *founders* will be the root nodes of our belief networks. We assume that the founders are members of a population in Hardy-Weinberg equilibrium (this is a conventional assumption). This yields the following table of priors:

$$p = p(a1)$$
$$q = p(a2) = 1 - p$$
$$p(a1a1) = p^2$$
$$p(a1a2) = 2pq$$
$$p(a2a2) = q^2$$

Finally, we need the *penetrance probabilities*. These are the probabilities of an individual having some phenotype, conditioned on its genotype. We give a set of penetrance probabilities for a recessive trait in Fig. 12.

The rules necessary to describe this situation are as follows:

Rule 5 *Transmission of Genotypes:*

```
(→ (child ?c ?p1 ?p2)
   (and (genotype ?c) :label ??CHILD-NODE
        (genotype ?p1) :label ??PARENT-1
        (genotype ?p2) :label ??PARENT-2)
   :prob ((??PARENT-1 ??PARENT-2 ⇒
              ??CHILD-NODE)
     ((a1a1 | a1a1 a1a1) 1)
     ((a1a1 | a1a1 a1a2) 0.5) ((a1a2 | a1a1 a1a2) 0.5)
     ((a1a2 | a1a1 a2a2) 1)
     ((a1a1 | a1a2 a1a1) 0.5) ((a1a2 | a1a2 a1a1) 0.5)
     ((a1a1 | a1a2 a1a2) 0.25) ((a1a2 | a1a2 a1a2) 0.5)
     ((a2a2 | a1a2 a1a2) 0.25)
     ((a1a2 | a1a2 a2a2) 0.5) ((a2a2 | a1a2 a2a2) 0.5)
     ((a1a2 | a2a2 a1a1) 1)
     ((a1a2 | a2a2 a1a2) 0.5) ((a2a2 | a2a2 a1a2) 0.5)
     ((a2a2 | a2a2 a2a2) 1))))
```

Rule 6 *Penetrance:*

```
(→ (phenotype ?NAME ?CONDITION)
   (→← (penetrance-prob ?PROB)
     (and (genotype ?NAME) :label ??GENOTYPE
          (phenotype ?NAME)
                :label ??PHENOTYPE))
     :prob ((??GENOTYPE ⇒ ??PHENOTYPE)
       ((present | a1a1) ?PROB)
       ((absent | a1a2) 1)
       ((absent | a2a2) 1)))))
```

Rule 7 *Penetrance Probability:*

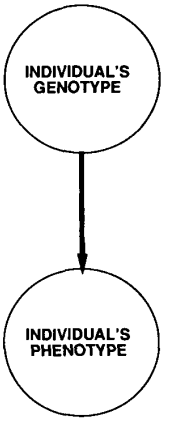
```
(penetrance-prob 1)
```

This statement will be examined by the rule for penetrance given above.

Rule 8 *Predicate Definitions:*

```
(defpreddist genotype hardy-weinberg simple-pform)
(defpreddist phenotype nil simple-pform)
```

The defpreddist forms given here assign the prior function of `genotype` nodes to be a function that will yield a Hardy-Weinberg distribution. Neither `genotype` nor `phenotype` need a causal combination function since nodes of this type will have only one pform. The function `simple-pform` simply checks to make sure a node has only one pform and, if so, carries out the appropriate action, reporting an error if not.



	Genotype		
Genotype	a1a1	a1a2	a2a2
:present	1	0	0
:absent	0	1	1

Fig. 12. Conditional probability matrix for penetrance probabilities.

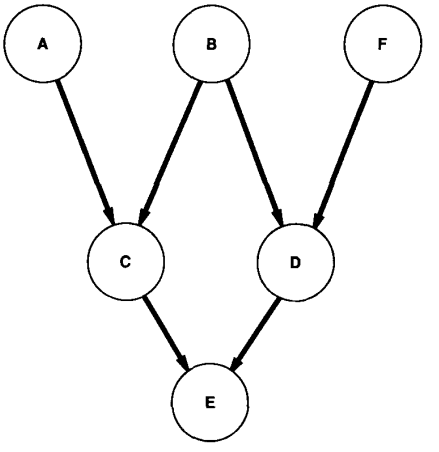


Fig. 13. Belief network/pedigree corresponding to a case of incest.

`phenotype` nodes have no prior function because they will never appear in isolation; they will only appear as children of `genotype` nodes.

We should note a couple of features of the above rules. First of all, note that we can use the same rules for many different values of p and q, simply by changing the prior function for genotype nodes. Furthermore, we can change the penetrance probability just by changing rule 7; therefore, this set of rules will work for any recessive condition. A different penetrance rule would be needed for a dominant condition, but it would be a straightforward variant of rule 6.

In [31], Jensen *et al.* give an example of a belief network for an example of incest. It is given as Fig. 13. Individual E is known to have the condition. We would create a diagram corresponding to this situation by asserting the following statements into a database containing the above rules:

```
(child C A B)
(child D B F)
```

(child E C D)
(phenotype E)

We would use the function `add-evidence` (not previously discussed) to record the fact that E has the condition:

```
(add-evidence (phenotype E) :present)
```

We could now evaluate the network to determine the likely genotypes of the members of this family.

Let us consider a case with a bit more motivation—the problem of predicting a future child's phenotype. Imagine that this is a pedigree for a family of animals and that one is deciding whether or not to breed healthy individuals C and D. Further imagine that you know that B has this dread condition but that you are unable to observe A and F. One would assert the same family-tree statements as above:

(child C A B)
(child D B F)
(child E C D)

One would also record the observations about the conditions of the various individuals:

(phenotype B)
(add-evidence (phenotype B) :present)
(phenotype C)
(add-evidence (phenotype C) :absent)
(phenotype D)
(add-evidence (phenotype D) :absent)

Now, because one is concerned about the outcome of breeding C and D, one would add the random variable

```
(phenotype E)
```

Then, one could evaluate the belief network, examine the distribution of `(phenotype E)`, which is the random variable representing the health of a hypothetical child of C and D, and on that basis decide one's breeding strategy.

IX. Dynamic Belief Networks in Natural Language Processing

In this section, we describe our application of dynamically constructed belief networks to the problem of natural language processing. This section of the paper may be of interest as far as providing a further example of network construction and one requiring more complex rules. However, it is completely self-contained and may safely be skipped.

Our interest in belief networks has grown out of our work on story understanding. The problem of story understanding is to take a story written in some natural language and "understand" it in the sense that one's program could answer questions about it. It is generally understood that this involves translating the story into an internal representation that will support the inferences necessary to answer questions. This internal representation should be an unambiguous representation of the meaning of the text. In this translation process, the program will make use of knowledge about language and about the world the language is being used to describe. Central to the problem of story understanding is plan recognition: the ability to recognize the plans and goals of characters in a story based on their actions.

For some time, we have been interested in the problems posed by uncertainty in story understanding. Uncertainty in language understanding comes from the ambiguity that exists at all levels of natural language: syntactic, semantic, and pragmatic. Our current program, Wimp3, operates by turning problems of text interpretation into probability questions such as "What is the probability that this word has this sense, given the evidence?" (where the evidence is the text of the story so far), and "What is the probability that the action described by the word plays a role in the actor's plan to achieve some known goal?" Wimp3 does this on a word-by-word basis. These probability questions are formulated as belief networks.

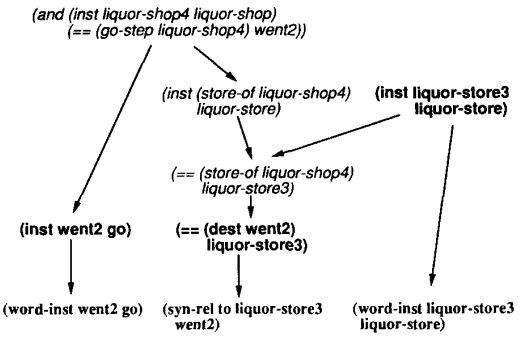


Fig. 14. Fragment of the belief network representation of the sentence "Jack went to the liquor -store."

The networks we create contain nodes describing the text at many levels of detail (see Fig. 14 for an example). At the leaves are the nodes describing the "evidence" we have observed: the input string. Above these are nodes describing the denotations of the words in the input (e.g., **liquor-store3** in Fig. 14). At the top of the networks are descriptions of the plans of the characters in the story (*liquor-shop4* is a hypothesized plan to buy liquor at a liquor store).

More precisely, Wimp3 works as follows:

1) A parser reads a single word of the English text. It produces statements that describe the words in the story and the syntactic relations between them.
2) The output of the parser is used by the network construction component. This component contains rules for language abduction.[2] It builds a net or extends the current net if some input has already been received.
3) We collect the portion of the belief network that may be affected by the most recent additions to the belief network. This subgraph is evaluated by a network-evaluation component.
4) If certain conclusions are overwhelmingly favored, they may be accepted as true to simplify further computation. Statements not rejected as too improbable will trigger further network expansion. If nodes are added to the network, go to step 3, else, return to step 1. This aspect of the work has not been thoroughly worked out. At the moment, we simply commit to beliefs based on the probability a statement has after a fixed period of time.

[2] *Abduction* is the process of reasoning to a best explanation, which is usually considered to involve reasoning from statements of the form "α causes β" and "β" to "α." For a discussion of abductive reasoning, see Ch. 8 of [11].

To return to the example of Fig. 14, on the first iteration after reading "liquor-store," we will add the statement `(inst liquor-store3 liquor-store)` to the database. This hypothesis will not be eliminated as false when the network is evaluated; therefore, we will consider it further, adding the nodes for a liquor-shopping plan. We do this iterative expansion of the network to keep its size small. For example, one possible (although unusual) use of the word "went" is to mean "died," as in "Franco went at 10 o'clock last night." In the context of "Jack went to the liquor-store," this sense is clearly ruled out by syntactic information. By expanding the belief network incrementally, we avoid adding nodes to represent explanations for the nonexistent event of Jack dying.

X. Related Work

There is a growing body of work concerned with knowledge-based model construction. The work most closely related to ours is that of Breese and some more limited work by Horsch and Poole. There are also a number of projects based on adaptations of deKleer's ATMS. For a more thorough survey, see [48]. More details on our work on language understanding is given in [9], [3], [8], [6], [5], [26], and [24].

In work done simultaneously with ours, Breese has developed very similar techniques for rule-based construction of probabilistic models[2]. The differences between his work and ours arise from the different problems we are trying to solve. Breese's method is aimed at creating belief networks and influence diagrams in order to reply to queries. Because our work is aimed at interpretation and requires complex hypotheses, we do not have clearly goal-directed networks.

Another difference is that Breese does not have anything corresponding to our causal combination functions. Fully fleshed out conditional probability distributions are associated with his network-constructing rules. When more than one such distribution applies, his system chooses which to apply according to some simple heuristics. We need to be able to specify in greater detail how to compare different probabilistic dependencies because our models are too big to describe more explicitly and because nodes and arcs are added to and deleted from our networks.

Two examples will clarify this important difference: First of all, consider our rule for finding word senses (rule 2). If each word has a large number of possible senses, it is clearly impossible for us to store a full conditional probability matrix for each word, given all its senses. This is impossible and unnecessary because the different causes interact in a stereotyped way. Second, our interpretation problem is hierarchical. We make hypotheses about actions and events to explain the text and then consider the plans and goals of characters in the story in order to explain these actions and events. We do not want to expand our networks to include plans hypothesized to explain the actions of characters until we are at least reasonably sure that the actions we are trying to explain actually occurred—that we have not simply misunderstood some verb. This leads to our need to make root nodes into interior nodes.

Work by Horsch and Poole, which was developed independently and is contemporary with our own, is similar to Breese's work but restricted to belief networks [29].

There has been a great deal of work on probabilistic ATMS's for dynamically constructing probabilistic models. An ATMS [19] is a multiple-context truth maintenance system. Laskey and Lehner [33] have shown that Dempster-Shafer theory can be mapped onto the ATMS in a straightforward way. D'Ambrosio developed a probabilistic ATMS [17], [18], and we developed a probabilistic ATMS tailored for language understanding [7].

In our experience, these logical-probabilistic hybrids are unsatisfactory for a number of reasons. Probabilistic inference and logical inference do not correspond perfectly. In systems based on logic, material implication is the most important operation, whereas in probabilistic systems, the most important inference is conditioning, which is not monotonic and is context dependent. This may be seen most clearly in D'Ambrosio's work. Conditioning requires keeping track of an amount of probabilistic "mass" lost when evidence is acquired (and renormalization done). This operation is difficult to do in an ATMS. Furthermore, logical inference does not permit us to distinguish between evidential and causal support for a proposition. In Pearl [39], the distinction between these two kinds of support is crucial for commonsense reasoning. Finally, from a pragmatic standpoint, working in an ATMS seemed wasteful to us since it required us to combine two expensive labeling procedures–ATMS context labeling and a further imposition of a probability measure.

XI. Possible Research Directions

One obvious way to extend the work done here would be to apply it to other graphical representations for probability problems. We have concentrated on belief networks because they are the most suitable to the natural language problems we face. For problems in decision theory, influence diagrams would be appropriate [40]. Indeed, Breese's network construction software works for influence diagrams as well as belief networks. It would be straightforward to extend FRAIL3 to allow description of influence diagrams.

Two other graphical representations suggest themselves as targets. The first is the Markov network—the undirected counterpart of belief networks. These are widely used in Bayesian approaches to pattern-recognition problems [23], [27]. Another possible representation is the chain graph, which is a class of graphs in which directed and undirected edges are mixed [22], [35].

Our NLP application has led us to consider graphical representations that include undirected edges. In the networks for language understanding, we have some nodes between which it is difficult to assign a causal direction. This occurs because in order to successfully process natural language requires at least two kinds of reasoning besides causal, diagnostic reasoning. One must be able to reason about identity relations among objects in order to understand the ways in which linguistic expressions refer to each other and to entities in the world. There are conclusions that may be drawn from

such coreference relations. However, these conclusions are not based on causality but rather on identity. One must also be able to perform taxonomic reasoning. Again, conclusions based on inheritance of features are not justified by causation.

Our techniques for specifying network topologies and indexing random variables would be just as applicable to Markov nets and chain graphs. The difficulties would arise in adapting the way the local probability distributions are specified. One of the greatest advantages of the belief network formalism is that it allows one to specify an entire probability distribution consistently, using only a limited number of conditional probabilities. It is more difficult to specify a consistent probability distribution over a undirected graph. The most common approach is to specify an energy function over states of the network by specifying a cost for various combinations of states. Unfortunately, these costs do not have any clear semantics. There are ways to translate a set of such conditional probabilities into a distribution over an undirected graph—that is, the basis of Lauritzen and Spiegelhalter's algorithm for evaluating belief networks—but this translation process makes use of the directed graph for assessment of probabilities. For a more thorough discussion of the advantages and disadvantages of Markov nets vis a vis belief networks, see ch. 3 of [39]. We find the chain graph formalism very attractive, but the independences expressed by a chain graph are not as well understood as are those of Markov nets or belief networks.

Another direction in which we could push the expressive capabilities of FRAIL3 is that of networks containing continuous as well as discrete random variables. What will be needed is a concise way to represent continuous distributions and an evaluation program that manipulates them effectively.

There are also some minor changes to the FRAIL3 language that might make it easier to read and use. In FRAIL3's present form, one must write prior and causal combination functions as lisp functions that read pforms and write IDEAL data structures. These functions are unnecessarily difficult to read and write. A more abstract syntax should be provided. As mentioned in Section V, we use $\rightarrow\leftarrow$ rules for two purposes: to incorporate additional information from background knowledge (e.g., find the possible set of meanings for a word) and to conditionalize network construction on the current state of the network. Conflating these two different tasks is probably not good software engineering. It would probably be better to distinguish these two cases syntactically.

Finally, in recent joint work with Breese, the first author has begun work on an architecture for KBMC that integrates the processes of model construction and evaluation [25].

XII. SUMMARY

In this paper we have outlined a language for constructing belief networks on an as-needed basis. This language of rules makes it convenient to define parameterized classes of probabilistic models. Such parameterized models are more convenient to use and more efficient to represent than full-blown belief networks. This language is designed to make it easy to take advantage of Pearl's suggestion that causes can be modeled as being combined in simple, stereotyped ways. We have demonstrated the usefulness of this technique with a simple example from genetics and an account of our work in language understanding.

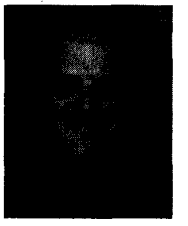
ACKNOWLEDGMENT

The authors would like to thank the reviewers for many helpful comments.

REFERENCES

[1] S. K. Andersen, K. G. Oleson, Kristian F. V. Jensen, and F. Jensen, "HUGIN—A shell for building Bayesian belief universes for expert systems," in *Proc. 11th Int. Joint Conf. Artificial Intell.*, 1989.
[2] J. S. Breese, "Construction of belief and decision networks," *Comput. Intell.*, vol. 8, no. 4, pp. 624–647, 1992.
[3] G. Carroll and E. Charniak, "A probabilistic analysis of marker-passing techniques for plan-recognition, in *Proc. Seventh Conf. Uncertainty Artificial Intell.* (Los Altos, CA), 1991, pp. 69–76.
[4] E. Charniak, "Bayesian networks without tears," *AI Mag.*, vol. 12, no. 4, pp. 50–63, 1992.
[5] E. Charniak and R. P. Goldman, "A Bayesian model of plan recognition," to be published in *Artificial Intell.*
[6] ——, "Probabilistic abduction for plan-recognition," in *Proc. Ninth Nat. Conf. Artificial Intell.*, 1991, pp. 160–165.
[7] ——, "A logic for semantic interpretation," in *Proc. Ann. Mtg. Assoc. Comput. Linguistics*, 1988, pp. 87–94.
[8] ——, "A semantics for probabilistic quantifier-free first-order languages, with particular application to story understanding," in *Proc. 11th Int. Joint Conf. Artificial Intell.* (Los Altos, CA), 1989, pp. 1074–1079.
[9] ——, "Plan recognition in stories and in life," *Uncertainty in Artificial Intelligence 5* (M. Henrion, R. D. Schachter, and J. F. Lemmer, Eds.). New York: Elsevier, 1990, pp. 343–351.
[10] E. Charniak and S. Husain, "A new admissible heuristic for minimal-cost proofs," in *Proc. Ninth Nat. Conf. Artificial Intell.*, 1991, pp. 446–451.
[11] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*. Reading, MA: Addison Wesley, 1985.
[12] R. M. Chavez and G. F. Cooper, "An empirical evaluation of a randomized algorithm for probabilistic inference," *Proc. Fifth Workshop Uncertainty Artificial Intell.* (Los Altos, CA), 1989, pp. 60–70.
[13] H. L. Chin and G. F. Cooper, "Stochastic simulation of Bayesian belief networks," in *Proc. Workshop Uncertainty Probability Artificial Intell.*, 1987, pp. 106–113.
[14] G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artificial Intell.*, vol. 42, nos. 2–3, pp. 393–405, 1990.
[15] ——, "NESTOR: A computer-based medical diagnosis aid that integrates causal and probabilistic knowledge," Ph.D. thesis, Stanford Univ., 1984.
[16] S. B. Cousins, W. Chen, and M. E. Frisse, *CABeN: A Collection of Algorithms for Belief Networks*, unpublished manual.
[17] B. D'Ambrosio, "A hybrid approach to reasoning under uncertainty," *Int. J. Approx. Rea.*, vol. 2, pp. 29–45, 1988.
[18] ——, "Process, structure, and modularity in reasoning with uncertainty," , *Uncertainty in Artificial Intelligence 4* (R. Shachter, T. Levitt, L. Kanal, and J. Lemmer, Eds.). Amsterdam: North Holland, 1990, pp. 15–25.
[19] J. deKleer, "An assumption-based TMS," *Artificial Intell.*, vol. 28, pp. 127–162, 1986.
[20] J. Doyle, "A truth maintenance system," *Artificial Intell.*, vol. 12, no. 3, pp. 231–272, 1979.
[21] E. Santos, Jr., "On the generation of alternative explanations with implications for belief revision," in *Proc. Seventh Conf. Uncertainty Artificial Intell.* (Los Altos, CA), 1991, pp. 339–347.
[22] M. Frydenberg, "The chain graph Markov property," Tech. Rep. 186, Dept. of Theoretical Stat., Inst. of Math., Univ. of Aarhus, 1989.
[23] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-6, pp. 721–741, 1984; republished in [42].
[24] R. P. Goldman, "A probabilistic approach to language understanding," Tech. Rep. CS-90-34, Comput. Sci. Dept., Brown Univ., 1990.
[25] R. P. Goldman and J. S. Breese, "Integrating model construction and evaluation," in *Uncertainty Artificial Intell. Proc. Eighth Conf.*, July 1992, pp. 104–111.

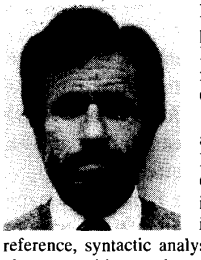
[26] R. P. Goldman and E. Charniak, "Probabilistic text understanding," *Stat. Comput.*, vol. 2, pp. 105–114, 1992.
[27] U. Grenander, "Tutorial in pattern theory," Tech. Rep., Div. of Applied Math., Brown Univ., Providence, RI, Dec. 1983.
[28] M. Henrion, "Propagating uncertainty by logic sampling in Bayes' networks," *Uncertainty in Artificial Intelligence* (J. F. Lemmer and L. F. Kanal, Eds.). Amsterdam: North-Holland, 1988, pp. 149–163, vol. 2.
[29] M. C. Horsch and D. A. Poole, "A dynamic approach to probabilistic inference using Bayesian networks," in *Proc. Sixth Conf. Uncertainty Artificial Intell.*, 1990, pp. 155–161.
[30] F. V. Jensen, "Bayesian updating in recursive graphical models by local computations," Tech. Rep. R 89-15, Inst. for Electron. Syst., Dept. of Math. Comput. Sci., Univ. of Aalborg, Aalborg, Denmark, 1989.
[31] F. V. Jensen, K. G. Oleson, and S. Anderson, "An algebra of Bayesian belief universes for knowledge-based systems," Tech. Rep. R 88-25, Inst. Electron. Syst., Dept. Math. Comput. Sci., Univ. of Aalborg, Aalborg, Denmark, 1988.
[32] J. H. Kim and J. Pearl, "A computation model for causal and diagnostic reasoning in inference systems," in *Proc. 6th Int. Joint Conf. Artificial Intell.* (Los Altos, CA), 1983, pp. 190–193.
[33] K. B. Laskey and P. E. Lehner, "Belief maintenance: An integrated approach to uncertainty management," in *Proc. Seventh Nat. Conf. Artificial Intell.*, 1988, pp. 210–214.
[34] S. L. Lauritzen and D. L. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *J. Royal Stat. Soc.*, vol. 50, pp. 157–224, 1988; republished in [42].
[35] S. L. Lauritzen, "Mixed graphical association models," Tech. Rep. R 88-21, Inst. for Electron. Systems, Dept. of Math. Comput. Sci., Univ. of Aalborg, Aalborg, Denmark, 1988.
[36] D. McAllester, "Reasoning utility package user's manual, Version one," Tech. Rep., Mass. Inst. Technol., Artificial Intell. Lab., 1982.
[37] R. E. Neapolitan, *Probabilistic Reasoning in Expert Systems*. New York: Wiley, 1990.
[38] P. Norvig, *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Los Altos, CA: Morgan Kaufmann, 1991.
[39] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Los Altos, CA: Morgan Kaufmann, 1988.
[40] R. D. Shachter, "Evaluating influence diagrams," *Oper. Res.*, vol. 34, no. 6, pp. 871–882, 1986; reprinted in [42].
[41] R. D. Shachter and M. A. Peot, "Simulation approaches to general probabilistic inference on belief networks," in *Proc. Fifth Workshop Uncertainty Artificial Intell.* (Los Altos, CA), 1989, pp. 311–318.
[42] G. Shafer and J. Pearl, Eds., *Readings in Uncertain Reasoning*. Los Altos, CA: Morgan Kaufmann, 1990.
[43] S. Shimony, "Algorithms for irrelevance-based partial MAPs," in *Proc. Seventh Conf. Uncertainty Artificial Intell.* (Los Altos, CA), 1991, pp. 370–377.
[44] S. Shimony and E. Charniak, "A new algorithm for finding MAP assignments to belief networks," in *Proc. Sixth Conf. Uncertainty Artificial Intell.*, 1990, pp. 98–103.
[45] D. J. Spiegelhalter, "Fast algorithms for probabilistic reasoning in influence diagrams with application in genetics and expert systems," *Influence Diagrams, Belief Nets and Decision Analysis* (R. M. Oliver and J. Q. Smith, Eds.). New York: Wiley, 1990, pp. 361–384, ch. 16 (Wiley Series on Probability and Mathematical Statistics); also in *Proc. Conf. Influence Diagrams Decision Anal. Inference Prediction* (Univ. of California, Berkeley), May 9-11, 1988.
[46] S. Srinivas and J. Breese, *IDEAL: Influence Diagram Evaluation and Analysis in Lisp*. Palo Alto, CA: Rockwell Int. Sci. Cent., May 1989.
[47] ——, "IDEAL: A software package for analysis of influence diagrams," *Proc. Sixth Conf. Uncertainty Artificial Intell.*, 1990, pp. 212–219.
[48] M. P. Wellman, J. S. Breese, and R. P. Goldman, "From knowledge bases to decision models," *Knowledge Eng. Rev.*, vol. 7, no. 1, pp. 35–53, 1992.



Robert P. Goldman received the B.A. degree in philosophy from Yale University, New Haven, CT, in 1983 and the M.Sc. and Ph.D. degrees in computer science from Brown University, Providence, RI, in 1988 and 1991, respectively. His thesis work concerned the application of Bayesian influence to text understanding.

Since July 1990, he has been a member of the faculty of the Computer Science Department of Tulane University, New Orleans, LA. In the summer of 1992, he was a visiting scientist at Rockwell International's Palo Alto Laboratories. His research interests include natural language processing and the application of Bayesian decision theory to artificial intelligence problems.

Dr. Goldman is a member of the IEEE Computer Society, the American Association for Artificial Intelligence, the Association for Computational Linguistics, and the Association for Computing Machinery.



Eugene Cherniak received the A.B. degree in physics from the University of Chicago, Chicago, IL, and the Ph.D. degree in computer science from the Massachusetts Institute of Technology, Cambridge.

He is currently Professor of Computer Science and Cognitive Science at Brown University, Providence, RI. He is Chairman of the Departmentof Computer Science at Brown. His research has been in the area of language understanding (paticularly in the resolution of ambiguity in noun-phrase reference, syntactic analysis, case determination, and word-sense selection), plan recognition, and more generally, abduction. In the last few years, his work has concentrated on the use of probabilistic methods to elucidate these problems. Most recently, he has become interested in statistical approaches to language learning. He has published three books: *Computational Semantics* with Y. Wilkes (1976), *Artificial Intelligence Programming* (now in a second edition) with C. Riesbeck, D. McDermott, and J. Meehan (1980, 1987), and *Introduction to Artificial Intelligence* with D. McDermott (1985).

He is a Fellow of the American Association of Artificial Intelligence and was previously a Councilor of the organization. He is on the editorial boards of the journals *Cognitive Science* (of which he was a founding editor) and *Computational Linguistics*.